Processes

Linux Sysadmin Decal, Fall 2020

Ryan Chan <rrchan>

What is a process?

- A process is an instance of a program being executed
- Has a **PID** - Process ID, a numerical identifier assigned to it
- Anything that you would think of as ‘running’ is *probably* a process
    - Or in some cases, a service or a collection of processes - more on that next lecture
- Ex. Bash, vim, ping

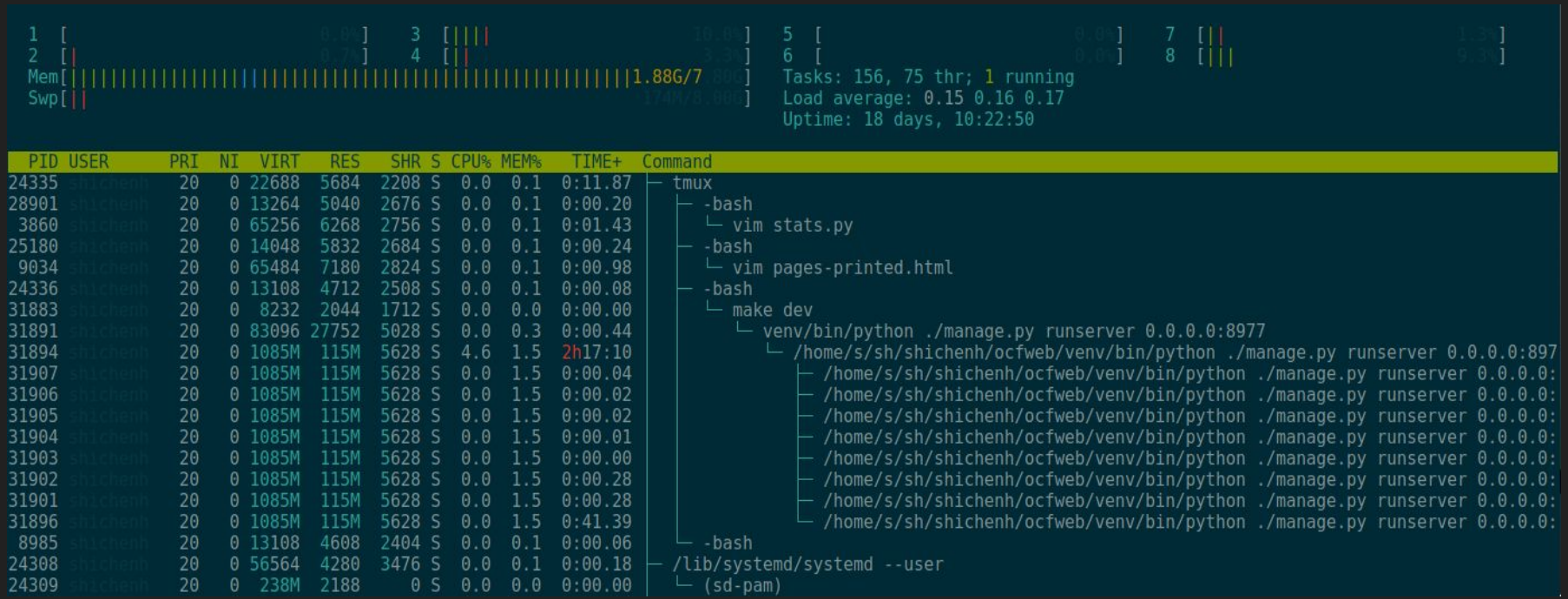
Where do processes come from?

- Processes come from other processes!
  - When a process creates another process, we call the originator the **parent** process, and the newly created process the **child** process.
  - This creates a tree-like hierarchy among processes
- Root of the process tree: **init**
  - First process run on startup, started by the kernel
  - PID 1
  - Different init systems exist. eg. You might see **systemd** as PID 1 on a system, depending on the distro you're using

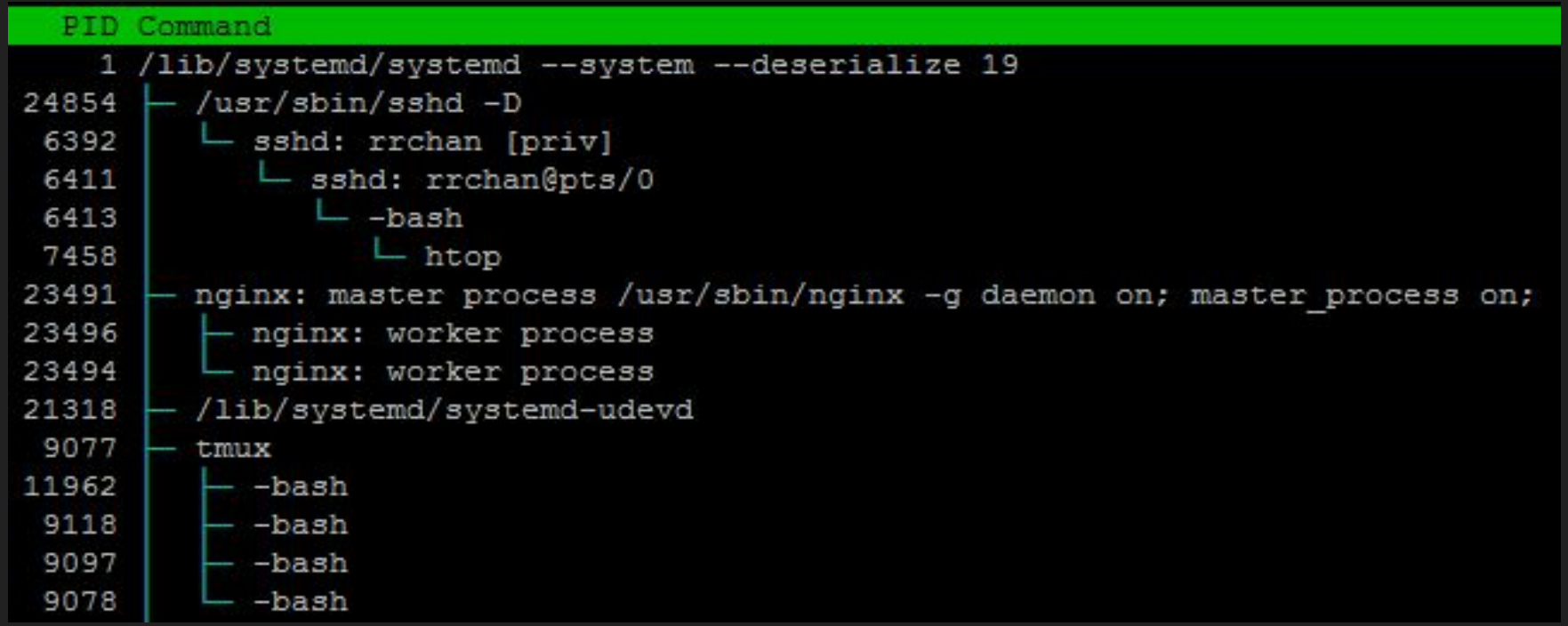
Some commands

- ps: print running processes
  - Default: print processes in your current session
  - -e: print all processes
  - -f: print with full information
  - -H: print a tree
  - -u (--user): print a certain user's processes
- kill, pkill: send a signal to a process (to end it)
- pgrep: search for a process by its name
- htop: show live process stats in a nice graphical format

htop


1 [0.0%] 3 [|||| 10.0%] 5 [0.0%] 7 [|| 1.3%]
2 [| 0.7%] 4 [|| 3.3%] 6 [0.0%] 8 [||| 9.3%]
Mem[|||||||||||||||||||||||||||||||||||||||||||||||||1.88G/7.80G]
Swp[|| 174M/8.00G]
Tasks: 156, 75 thr; 1 running
Load average: 0.15 0.16 0.17
Uptime: 18 days, 10:22:50
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
24335 shichenh 20 0 22688 5684 2208 S 0.0 0.1 0:11.87 ├─ tmux
28901 shichenh 20 0 13264 5040 2676 S 0.0 0.1 0:00.20 │ ├─ -bash
3860 shichenh 20 0 65256 6268 2756 S 0.0 0.1 0:01.43 │ │ └─ vim stats.py
25180 shichenh 20 0 14048 5832 2684 S 0.0 0.1 0:00.24 │ ├─ -bash
9034 shichenh 20 0 65484 7180 2824 S 0.0 0.1 0:00.98 │ │ └─ vim pages-printed.html
24336 shichenh 20 0 13108 4712 2508 S 0.0 0.1 0:00.08 │ ├─ -bash
31883 shichenh 20 0 8232 2044 1712 S 0.0 0.0 0:00.00 │ │ └─ make dev
31891 shichenh 20 0 83096 27752 5028 S 0.0 0.3 0:00.44 │ │ └─ venv/bin/python ./manage.py runserver 0.0.0.0:8977
31894 shichenh 20 0 1085M 115M 5628 S 4.6 1.5 2h17:10 │ │ └─ /home/s/sh/shichenh/ocfweb/venv/bin/python ./manage.py runserver 0.0.0.0:897
31907 shichenh 20 0 1085M 115M 5628 S 0.0 1.5 0:00.04 │ │ ├─ /home/s/sh/shichenh/ocfweb/venv/bin/python ./manage.py runserver 0.0.0.0:
31906 shichenh 20 0 1085M 115M 5628 S 0.0 1.5 0:00.02 │ │ ├─ /home/s/sh/shichenh/ocfweb/venv/bin/python ./manage.py runserver 0.0.0.0:
31905 shichenh 20 0 1085M 115M 5628 S 0.0 1.5 0:00.02 │ │ ├─ /home/s/sh/shichenh/ocfweb/venv/bin/python ./manage.py runserver 0.0.0.0:
31904 shichenh 20 0 1085M 115M 5628 S 0.0 1.5 0:00.01 │ │ ├─ /home/s/sh/shichenh/ocfweb/venv/bin/python ./manage.py runserver 0.0.0.0:
31903 shichenh 20 0 1085M 115M 5628 S 0.0 1.5 0:00.00 │ │ ├─ /home/s/sh/shichenh/ocfweb/venv/bin/python ./manage.py runserver 0.0.0.0:
31902 shichenh 20 0 1085M 115M 5628 S 0.0 1.5 0:00.28 │ │ ├─ /home/s/sh/shichenh/ocfweb/venv/bin/python ./manage.py runserver 0.0.0.0:
31901 shichenh 20 0 1085M 115M 5628 S 0.0 1.5 0:00.28 │ │ ├─ /home/s/sh/shichenh/ocfweb/venv/bin/python ./manage.py runserver 0.0.0.0:
31896 shichenh 20 0 1085M 115M 5628 S 0.0 1.5 0:41.39 │ │ └─ /home/s/sh/shichenh/ocfweb/venv/bin/python ./manage.py runserver 0.0.0.0:
8985 shichenh 20 0 13108 4608 2404 S 0.0 0.1 0:00.06 │ └─ -bash
24308 shichenh 20 0 56564 4280 3476 S 0.0 0.1 0:00.18 ├─ /lib/systemd/systemd --user
24309 shichenh 20 0 238M 2188 0 S 0.0 0.0 0:00.00 │ └─ (sd-pam)

More htop


PID Command
1 /lib/systemd/systemd --system --deserialize 19
24854 ├─ /usr/sbin/sshd -D
6392 │ └─ sshd: rrchan [priv]
6411 │ └─ sshd: rrchan@pts/0
6413 │ └─ -bash
7458 │ └─ htop
23491 ├─ nginx: master process /usr/sbin/nginx -g daemon on; master_process on;
23496 │ ├─ nginx: worker process
23494 │ └─ nginx: worker process
21318 ├─ /lib/systemd/systemd-udevd
9077 ├─ tmux
11962 │ ├─ -bash
9118 │ ├─ -bash
9097 │ ├─ -bash
9078 │ └─ -bash

Signals: Controlling Processes

- A means of sending messages to other processes
- **SIGTERM** (15): tell a process to exit now
- **SIGINT** (2): the signal sent when you press Ctrl+C
- **SIGHUP** (1): the signal sent when the user closes the terminal window
- **SIGCHLD**: tell a process to collect child data
- **SIGKILL** (9): kill a process without giving it the chance to clean up
- **SIGSTOP** (varies; 19 on GNU/Linux): suspend a process (Ctrl + Z)
- **SIGCONT** (varies; 18 on GNU/Linux): resume a process
- The signals **SIGKILL** and **SIGSTOP** cannot be handled
- Can send using the kill, pkill, and killall commands
- e.g. kill -STOP 3276, kill -9 7283

Lifecycle of a process

1. Process is created, typically forked from a parent process
    a. Process is assigned a PID, operating system begins tracking process info
2. Process does whatever it's supposed to be doing
3. Process ends
    a. Operating system continues to hold onto resource info, PID, and termination status code for the process
    b. OS sends a **SIGCHLD** signal to the parent process, informing the parent that a child process has ended and that there is info to be collected
    c. Parent process collects the data by calling a *wait* function
    d. OS releases all remaining data about the process

Zombie processes

- What happens if a child process ends, but the parent function does not collect the data that the OS is keeping about the child?
  - Child process will remain in the process table with its PID and other info stored, indefinitely
  - Known as a **Zombie Process**
  - Child process has ‘died’, but the parent process has not fulfilled its responsibility to ‘reap’ the child
- Not a *huge* deal, memory impact is low

Orphan Processes

- What happens to the children of a parent process if the parent process ends before its children do?
  - Child Processes are reparented, or adopted, by PID 1
  - Known as **Orphan Processes**
- What happens to processes that are both zombies and orphans?
  - Still gets sent to PID 1
  - init systems will periodically clean up zombies that are children of it

Process IO

- Every process on a Unix system initialized with three open file handles:
- Often referred to as streams
  1. **stdin** (fd 0): where a process looks for input
  2. **stdout** (fd 1): where a process can write output
  3. **stderr** (fd 2): where a process can write errors

Review: Process IO Examples

- $ echo “stdout is the terminal”
- $ echo “stdout is the terminal” >&1
- $ echo “writing to stderr, also the terminal!” >&2
  - >&#: also send to the specified stream
- $ ./program.sh 1>outfile 2>errfile
- $ ./program.sh 2>&1
  - Send stderr to wherever stdout is going
- stdin, new way to use the cat command?
  - $ cat < somefile.txt

Background Processes

- Noninteractive processes
- Started on the shell by appending an ‘&’ to the command
- One quirk: **stdout** and **stderr** inherited from parent shell, only **stdin** is detached
- These processes are still children of the current shell!
    - When a shell receives a **SIGHUP**, which is sent when you logout, it will also send a **SIGHUP** to each of its children, which will terminate them!

Some job control

- Jobs are child processes that a shell is taking responsibility for - not a universal Unix concept, just something that most shells do on their own
- **jobs**: See what child processes of the shell are running in the background
- **disown**: Removes a process from the shell's list of jobs
  - Will prevent the shell from sending the process a **SIGHUP**
  - The process will remain a child of the shell that ran it until the shell is closed, at which point the process will be **orphaned**
    - Gets sent to PID 1
  - Does not change the STDIN, STDOUT, or STDERR destinations of the process!
    - If one of the streams is pointing to the shell when it gets closed, the process will probably error out!

More job control

- Some more things you can do with jobs
- $ ping -c5 ocf.berkeley.edu
- Ctrl+Z to suspend the process
- See the process listed when you run **jobs**
- **bg** to have the process resume in the background
- **fg** to bring this to foreground

Advanced process management

- Disowning background processes has its limits! What if have a process that I want running in the background all the time, and I want more tools to actually manage it?

Find out next time, in lecture b7 - Services!